Tagungsband 7. E.I.S.-Workshop, TU-Chemnitz-Zwickau, 7./8. November 1995, pp. 182–191

Layoutgenerierung mit genetischen Algorithmen*

Volker Schnecke, Oliver Vornberger
Universität Osnabrück
Fachbereich Mathematik/Informatik
49069 Osnabrück
{volker|oliver}@informatik.uni-osnabrueck.de
http://brahms.informatik.uni-osnabrueck.de/prakt/prakt.html

Zusammenfassung

Die Layout-Generierung im VLSI-Design-Zyklus stellt eine nahezu ideale Anwendung für genetische Algorithmen dar. Üblicherweise wird dieses Problem aufgrund seiner Komplexität in einer Reihe von aufeinanderfolgenden Teilproblemen bearbeitet, die jedoch wegen der starken Abhängigkeiten untereinander sinnvollerweise gleichzeitig optimiert werden sollten. Ein genetischer Algorithmus ist in der Lage, durch eine hierarchische Plazierung generierte Layouts global zu optimieren, wobei zuvor getroffene lokale Entscheidungen bezüglich der Verdrahtung mitberücksichtigt werden.

1 Einleitung

Die meisten der kombinatorischen Optimierungsprobleme, welche in den unterschiedlichsten Anwendungen auftreten, sind trotz der heutzutage verfügbaren, gigantischen Rechenleistung zu komplex, um direkt und exakt gelöst zu werden. Abhilfe schaffen hier eine Aufteilung in kleinere Teilprobleme und die Anwendung von heuristischen Lösungsverfahren. Ein typisches Beispiel für ein derartiges Problem ist das Design von Macro-Cell Layouts, was in der Regel in einer Folge von Teilaufgaben (Plazierung, Verdrahtung, Kompaktierung) bearbeitet wird. Im folgenden wird ein genetischer Algorithmus vorgestellt, welcher eine Reihe dieser üblichen Teilaufgaben gleichzeitig löst. Obwohl genetische Algorithmen ein sehr mächtiges Optimierungsverfahren darstellen, wurden sie in der Vergangenheit immer nur auf Teilprobleme aus dem Bereich des physikalischen Designs von VLSI-Layouts angewandt. So lösen z. B. Cohoon *et. al.* [1, 2] das Floorplanning Problem mit genetischen Algorithmen, wobei bei der Plazierung auch der globale Verdrahtungsaufwand berücksichtigt wird. Einen Algorithmus zur Durchführung der detaillierten Kanalverdrahtung beschreiben Lienig und Thulasiraman [5]. Esbensen [4] befaßt sich sowohl mit der Plazierung, als auch mit der Optimierung der globalen Verdrahtung, allerdings in getrennten Schritten. Wenn man jedoch von dem herkömmlichen,

*Diese Arbeit wird gefördert im Rahmen des BMBF-Verbundprojektes „HYBRID – Anwendungen paralleler genetischer Algorithmen in der kombinatorischen Optimierung“.

seriellen Lösungsweg Abstand nimmt, ist es möglich, die detaillierte Verdrahtung während der Plazierung durchzuführen und die Optimierung der globalen Verdrahtung einem genetischen Algorithmus zu überlassen.

2 Macro-Cell Layout Design

Die Problemstellung beim Design von *Macro-Cell Layouts* besteht darin, eine Menge von rechteckigen Modulen (Blöcken) auf einer minimalen, rechteckigen Fläche zu plazieren und zu verdrahten. Die Module können dabei an beliebigen Positionen innerhalb des Layouts angeordnet werden. Oft sind deren Dimensionen während der Plazierung noch nicht endgültig festgelegt, d. h. es existieren unterschiedliche Layout-Alternativen, welche durch Form-Funktionen *(shape-functions)* beschrieben werden. Diese Form-Funktionen sind Treppenfunktionen, die durch eine Liste von einigen (minimalen) Implementierungen für ein Modul spezifiziert werden [7].

Die Lösung dieses hochgradig komplexen kombinatorischen Optimierungsproblems erfolgt üblicherweise in mehreren aufeinanderfolgenden Phasen [8, 10]: Zunächst wird die Plazierung aller Module unter Berücksichtigung des voraussichtlich benötigten Verdrahtungsplatzes durchgeführt, danach die globalen Verdrahtungswege bestimmt und anschließend die detaillierte Verdrahtung in den Kanälen zwischen den Modulen ermittelt. Da in der ersten Phase im allgemeinen zuviel Verdrahtungsplatz reserviert wird, muß das Layout anschließend noch kompaktiert werden.

3 Genetische Algorithmen

Genetische Algorithmen sind ein heuristisches Lösungsverfahren, welches nach dem Prinzip der Evolution arbeitet [6]. Die Evolution ist der kontinuierliche Prozeß, der seit Anbeginn der Zeit zum Artenwandel und zur Bildung neuer Arten und Organisationstypen allen Lebens geführt hat. Es ist ein Optimierungsprozeß, dessen Erfolg jedoch nicht quantitativ meßbar ist. Die Erfolge der Evolution drücken sich in „Anpassungsfähigkeit“ und „Überlebensfähigkeit“ aus. Es ist also keine zielgerichtete Optimierung, sondern ein ständiger Anpassungsprozeß an die sich ändernden Lebensbedingungen. Ein Kernbegriff der Evolutionstheorie ist das Prinzip „Survival of the fittest“, welches von Charles Darwin in seinem Klassiker [3] eingeführt wurde. Nach diesem Prinzip arbeiten die genetischen Algorithmen, indem sie den Evolutionsprozeß simulieren.

Wie in der Natur auch, startet der Evolutionsprozeß mit einer Menge von einzelnen *Individuen*, der sogenannten *Population*. Die Individuen sind komplette Lösungen für ein Optimierungsproblem, die durch die Zuordnung einer Qualität, der sogenannten *Fitneß*, bewertet werden. Aus den besten dieser Elemente werden im Laufe einer *Generation* durch Anwendung der genetischen Operatoren *Mutation* und *Crossover* neue Lösungen erzeugt, welche am Ende der Generation andere Individuen in der Population ersetzen, falls die Qualität der neu gene-

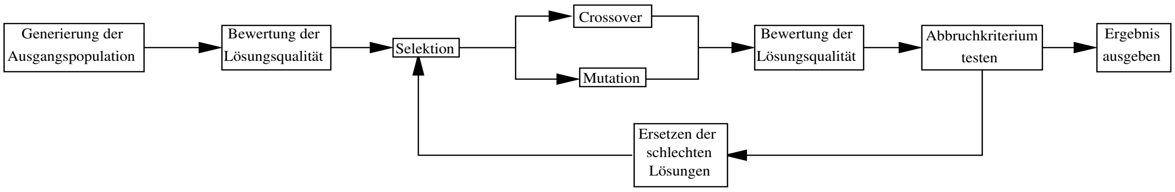

Abbildung 1: Der Ablauf eines genetischen Algorithmus

rierten Lösungen besser ist (siehe Abb. 1). Der Mutations-Operator kopiert ein Individuum und führt an der Kopie zufällige Änderungen durch. Der Crossover-Operator entspricht der Fortpflanzung in der Biologie, d. h. es werden zwei „Eltern“ aus der Population ausgewählt *(selektiert)*, aus denen ein neues Individuum *(Offspring)* generiert wird, welches Eigenschaften von beiden *Parent-Individuen* erbt. Der Crossover-Operator steuert den Optimierungsprozeß, indem er gezielt gute Lösungen kombiniert, während der Mutationsoperator die Varianz in der Population erhöht und somit die Konvergenz in ein lokales Optimum verhindert.

3.1 Die Kodierung für ein Layout

In Anlehnung an die Biologie unterscheidet man die *Genotyp-* und die *Phenotyp*-Darstellung eines Individuums. Der Genotyp ist die Kodierung, auf der die genetischen Operatoren arbeiten, und der die gesamte genetische Information eines Individuums enthält, während der Phenotyp die „physikalische“ Ausprägung des Individuums festlegt. Die Genotyp-Kodierung besteht in der klassischen Definition der genetischen Algorithmen aus einzelnen *Genen*, die unterschiedliche Werte, die sogenannten *Allele* annehmen können. Die einfachste Kodierung eines Genotypen ist ein Bitstring, wobei jedes Bit ein Gen darstellt, die Allelen sind 0 und 1. Für die Optimierung von kontinuierlichen Zielfunktionen ist der Genotyp ein Vektor von Gleitkommazahlen. Die genetischen Operatoren für diese Fälle verändern ein oder mehrere Gene (Mutation), oder kreuzen zwei Gen-Strings, indem jedes Gen des Offsprings eines der beiden möglichen Allele aus dem entsprechenden Gen eines Parents übernimmt (Crossover).

Für ein diskretes Optimierungsproblem wie die Layoutgenerierung kann die Genotyp-Kodierung nicht wie in der Biologie als ein String von Genen erfolgen. Um die Lage der Module auf dem Layout – also den Floorplan – zu beschreiben, ist ein binärer Schnittbaum geeignet (siehe Abb. 2). Die Blätter dieses Baumes beschreiben die Module *(Blöcke)* des Schaltkreises, und die inneren Knoten repräsentieren Teil-Layouts *(Meta-Blöcke)*. Da der Schnittbaum nur die relative Position der Blöcke beschreibt, werden den inneren Knoten noch zusätzliche Informationen bezüglich der Anordnung der zusammengefaßten Blöcke, der Verdrahtung und der möglichen Layout-Alternativen (über Form-Funktionen) zugeordnet. Existieren für einen Block mehrere Implementierungen, so ergeben sich für alle Meta-Blöcke, welche diesen Block enthalten, mehrere Layout-Alternativen, die durch Form-Funktionen für die Meta-Blöcke

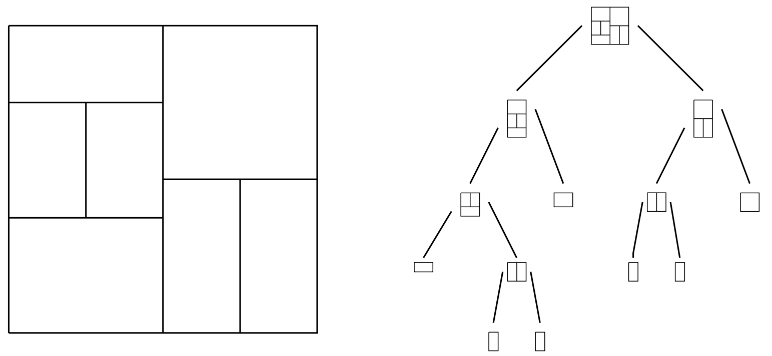

Abbildung 2: Ein Floorplan und die Repräsentation als binärer Schnittbaum

gespeichert werden. Dieses führt zu einer deutlich schnelleren Konvergenz des genetischen Algorithmus, da bei der Konstruktion eines Individuums bzw. nach der Anwendung eines genetischen Operators immer die global günstigste Layout-Alternative für jedes einzelne Modul ausgewählt werden kann [9].

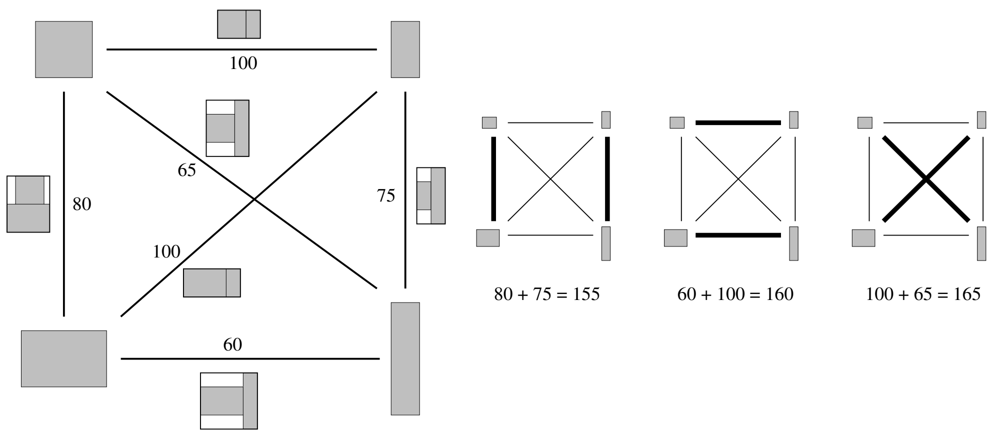

Abbildung 3: Ein Graph für vier Blöcke und die drei möglichen Matchings

Die Konstruktion des Schnittbaumes für ein Individuum der Ausgangspopulation erfolgt „bottom-up“, d. h. es werden zunächst jeweils zwei Elementar-Blöcke zu einem Meta-Block zusammengefaßt und im folgenden jeweils zwei Meta-Blöcke vereinigt, bis der Schnittbaum vollständig ist. Dabei werden die Blöcke nicht zufällig gepaart, sondern mit Hilfe eines *ite-*

rierten Matching-Verfahrens Individuen erzeugt, die dicht gepackte Meta-Blöcke enthalten. Hierzu wird ein vollständiger Graph konstruiert, dessen Knoten die Blöcke repräsentieren und dessen Kanten mit einem Wert gewichtet sind, der die Qualität des Meta-Blockes beschreibt, der entsteht, wenn die beiden zu einer Kante adjazenten Blöcke zusammengefaßt würden (siehe Abb. 3).

Ein *Matching* in dem Graphen ist eine Menge von disjunkten Knotenpaaren, wobei die gepaarten Knoten durch eine Kante verbunden sind. Es wird in dem Graphen ein maximal gewichtetes Matching *(maximum weight matching)* ermittelt, d. h. ein Matching, bei dem die Summe aller beteiligten Kantengewichte maximal ist. Dieses entspricht einer Menge von Meta-Blöcken mit maximaler Qualität, also mit einem minimalen Gesamtverschnitt. Das Matching wird in mehreren Iterationen durchgeführt, d. h. in der zweiten Iteration werden Meta-Blöcke, die aus jeweils zwei Elementar-Blöcken bestehen, gepaart und dieses Verfahren wird angewendet, bis nur noch zwei Meta-Blöcke vorhanden sind, die dann in der Wurzel des Schnittbaumes vereinigt werden.

3.2 Die Konstruktion der Verdrahtung

Im Gegensatz zu den üblichen Verfahren, wo Plazierung und Verdrahtung aufeinanderfolgende Prozesse darstellen, wird in dem genetischen Algorithmus bei der Konstruktion eines Meta-Blockes die komplette Verdrahtung innerhalb dieses Meta-Blockes durchgeführt (siehe Abb. 4). In der aktuellen Implementation wird dafür noch kein spezielles Verdrahtungsverfahren eingesetzt, es wir lediglich jedem Netz im Kanal zwischen den Modulen eine Spur zugewiesen. Netze, die noch weitere als die Terminals an einem der beiden beteiligten Blöcke verbinden, werden an die entsprechende Position auf einer der Außenwände des resultierenden Meta-Blockes weitervererbt. Es ist an dieser Stelle schon möglich, die detaillierte Verdrahtung

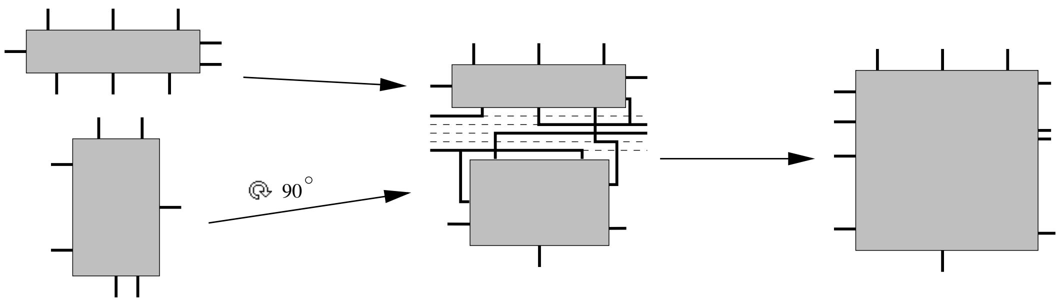

Abbildung 4: Die Durchführung der Verdrahtung bei der Konstruktion eines Meta-Blockes

festzulegen, da der Meta-Block in den höheren Ebenen des Baumes als fester Block angesehen wird, d. h. durch das durch ihn repräsentierte Teil-Layout werden später keine weiteren Netze hindurch geroutet.

Der Meta-Block, welcher durch die Wurzel des Schnittbaumes repräsentiert wird, besitzt an den Außenwänden Terminals, die entweder durch Drähte am äußeren Rand des Layouts verbunden oder an Pads angebunden werden, die die Ein-/Ausgabe übernehmen. Die Plazierung der Pads erfolgt im Anschluß an die Konstruktion des Schnittbaumes. Dabei werden die Pads um das Layout herum plaziert und die Netze – wiederum mit Hilfe eines Matching-Verfahrens – den Pads zugeordnet.

4 Die Optimierung durch den genetischen Algorithmus

Nachdem durch das iterierte Matching – zumindest in den unteren Ebenen des Schnittbaumes – dicht gepackte Meta-Blöcke generiert wurden und innerhalb der entsprechenden Teil-Layouts die Verdrahtung konstruiert wurde, ist es die Aufgabe des genetischen Algorithmus, die Layouts „global" zu optimieren. Dazu sind die vorhandenen Meta-Blöcke so umzuarrangieren, daß die Layout-Fläche minimal wird und möglichst optimale globale Verdrahtungswege entstehen. Dieses ist der wesentliche Unterschied zu dem herkömmlichen Ansatz im Layout-Design: Dort ist bei der Erstellung des Floorplans und der Konstruktion der globalen Verdrahtung jeweils eine globale Sicht auf die Anordnung der Module gegeben. Eben diese globale Sicht fehlt bei der Konstruktion eines Individuums. Hier werden lediglich lokale Entscheidungen getroffen und somit gute Teil-Layouts geschaffen, die globale Sicht auf das gesamte, sich entwickelnde Layout bleibt dem genetischen Algorithmus vorbehalten.

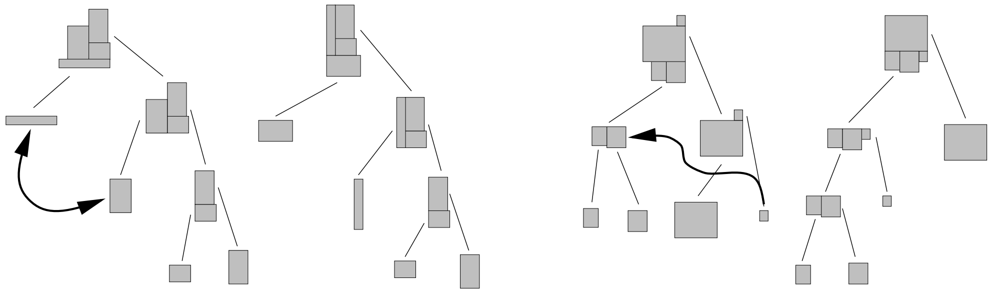

Abbildung 5: Mutation durch Austausch von Blöcken (links) oder durch Verändern der Struktur des Schnittbaumes (rechts)

Ein Mutationsoperator verändert zufällig ein Individuum, d. h. es geschieht nicht zielgerichtet und führt somit in den seltensten Fällen direkt zu einer Verbesserung (bzgl. der Fitneß des Individuums). Es existieren verschiedene Mutationsoperatoren, die mit unterschiedlicher

Häufigkeit angewendet werden. Ein Individuum wird durch Austausch von Knoten (Abb. 5, links) oder durch eine Veränderung der Struktur des Schnittbaumes (Abb. 5, rechts) mutiert. Andere Mutationsoperatoren verändern die Positionierung (aufeinander oder nebeneinander) und die Rotation der beiden Blöcke innerhalb des Meta-Blockes oder ändern die Seite, aus der ein Netz aus einem Kanal herausgeführt wird. Es ist notwendig, daß eine optimale Lösung aus einem beliebigen Layout nur durch Anwendung von Mutationsoperatoren erzeugt werden kann, denn durch die Anwendung des Crossover-Operators werden keine vollkommen neuen Individuen erzeugt, sondern nur Eigenschaften von schon existierenden Individuen vererbt.

Der Crossover-Operator wählt zwei Individuen aus, aus denen ein Offspring generiert wird. Diese Auswahl erfolgt nicht aus der gesamten Population, sondern nur aus der Menge der $T\,\%$ besten Individuen *(truncation selection)*. Als Maß für die Fitneß eines Individuums dient die Fläche des entsprechenden Layouts. In der aktuellen Implementation werden beim Crossover zwei disjunkte Meta-Blöcke aus beiden Parent-Individuen ausgewählt, welche die Grundlage für den Offspring bilden. Da es unwahrscheinlich ist, daß sich die beiden durch diese Meta-Blöcke repräsentierten Teil-Layouts zu einem vollständigen Layout ergänzen, müssen in dem Schnittbaum des Offsprings die noch fehlenden Blöcke ergänzt werden. Dabei wird wiederum das iterierte Matching angewandt, um die Verschnittfläche in diesem Teil des Layouts möglichst gering zu halten.

Die Wirkung des Crossover-Operators ist bei diskreten Optimierungsproblemen nur sehr gering, da bei der direkten Kreuzung nicht notwendigerweise korrekte Lösungen entstehen. Aus diesem Grund muß das Offspring-Individuum nach der Operation repariert werden, oder kann – wie in diesem Fall – nur teilweise durch Kreuzung zweier Individuen erzeugt werden. Hierdurch ist der Anteil der Vererbung deutlich geringer als in den meisten Anwendungen mit einer einfacheren Kodierung, da dort jedes Gen im Offspring-Genotyp direkt aus einem der beiden Parent-Individuen übernommen wird.

5 Ergebnisse

Der genetische Algorithmus wurde mit Layout-Benchmarks, welche ursprünglich vom MCNC (North Carolina's microelectronics, computing and networking center) für einen Workshop zur Verfügung gestellt wurden, getestet. Die Testinstanzen sind partitionierte Schaltkreise mit 10 bis 49 Modulen und 123 bis 408 Netzen. Abbildung 7 zeigt ein mit dem genetischen Algorithmus generiertes Layout für den Schaltkreis *ami33* mit 33 Modulen und 123 Netzen. Dieses Layout wurde in 500 Generationen berechnet, was bei einer Populationsgröße von 20 Individuen 10000 Operationen (Mutation bzw. Crossover) entspricht.

Die Entwicklung der Qualität des besten Individuums in der Population zeigt Abbildung 6. Dargestellt ist der Durchschnitt über jeweils 10 Läufe des genetischen Algorithmus für den Schaltkreis *ami49* mit 49 Modulen und 408 Netzen. Anhand dieser Grafik wird der Vorteil des iterierten Matchings bei der Konstruktion der Ausgangspopulation deutlich. Die oberen Kurven beschreiben die Qualität der Layouts für Optimierungen, die mit vollkommen

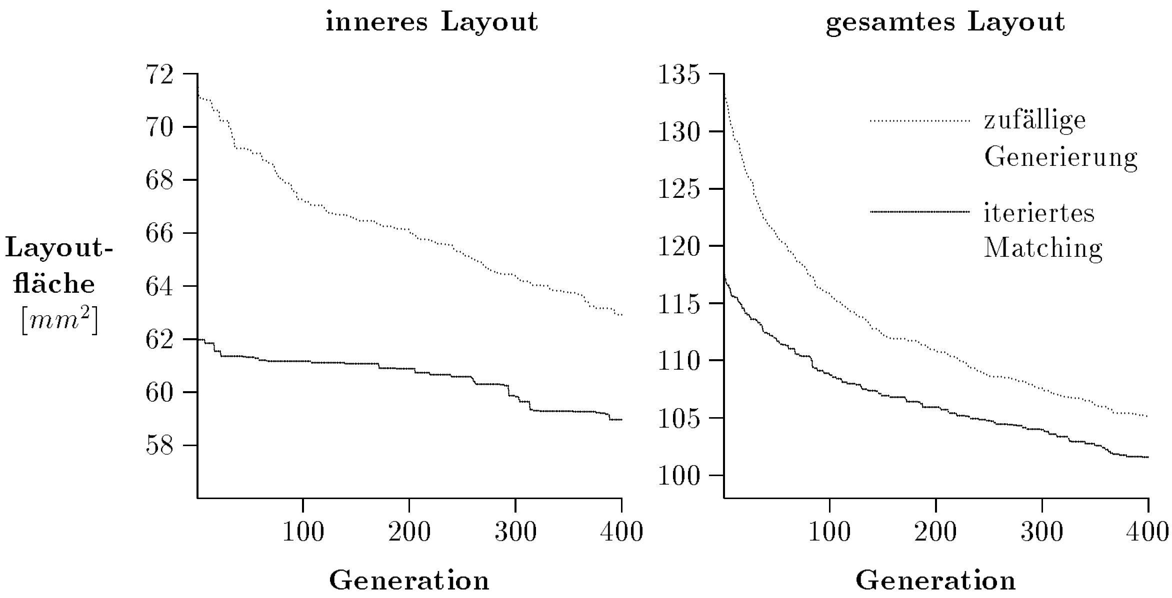

Abbildung 6: Die Entwicklung der Layoutfläche in Abhängigkeit von der Generierung der Ausgangspopulation

zufällig erzeugten Individuen starteten, während die durch die unteren Kurven dargestellten Daten von Läufen des genetischen Algorithmus stammen, bei denen alle Individuen mit Hilfe des iterierten Matchings erzeugt wurden. Neben der eigentlichen Fitneß, nämlich der Gesamtfläche (rechte Grafik), ist auch die Entwicklung des inneren Teils des besten Layouts gezeigt (linke Grafik), da sich nur hier das iterierte Matching direkt auswirkt. Die Größe des Verdrahtungsplatzes am Rand des Layouts, über den die Verdrahtung der Pads und die Vereinigung von Netzen mit Terminals an unterschiedlichen Seiten des Layouts realisiert wird, kann beim iterierten Matching nicht berücksichtigt werden, da dieser Platzbedarf erst nach der vollständigen Konstruktion des Schnittbaumes ermittelt werden kann. Bemerkenswert ist, daß die innere Layoutfläche in der mit dem Matching-Verfahren konstruierten Ausgangspopulation schon besser ist, als die entsprechenden Ergebnisse der zufälligen Läufe nach 400 Generationen.

6 Zusammenfassung und Ausblick

Das vorgestellte Verfahren optimiert mit Hilfe eines genetischen Algorithmus sowohl die Plazierung als auch die Verdrahtung bei der Layoutgenerierung. Möglich ist dieses durch eine geeignete Genotyp-Kodierung als binärer Schnittbaum, der generiert wird, indem Blöcke bzw.

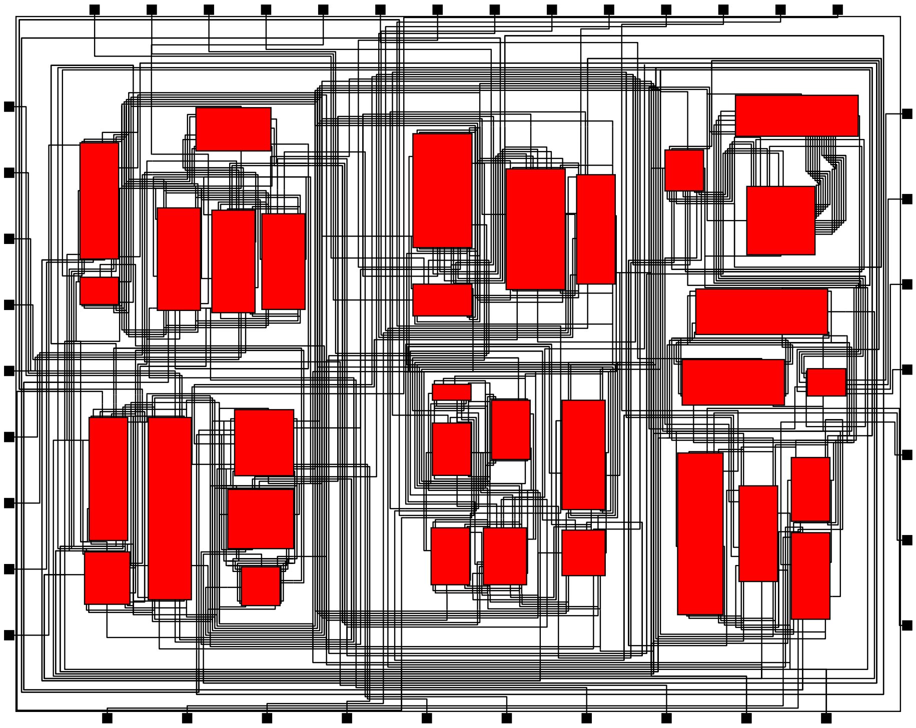

Abbildung 7: Ein Layout für eine Instanz mit 33 Blöcken und 123 Netzen

Teil-Layouts zu Meta-Blöcken und letztendlich zum Gesamtlayout zusammengefaßt werden. Bei der Konstruktion dieser Teil-Layouts wird die gesamte, innerhalb dieses Teils verlaufende Verdrahtung festgelegt. Der genetische Algorithmus permutiert die Teil-Layouts und ermittelt dadurch sowohl eine optimale globale Anordnung der Blöcke, als auch eine günstige Verdrahtung.

Bei der Konstruktion eines Teil-Layouts wird zur Zeit in dem Kanal zwischen zwei Blöcken für jedes Netz eine Spur reserviert, was in dem generierten Layout zu größeren Verschnittflächen führt. Eine Verbesserung wäre hier durch eine Kompaktierung im Anschluß an die Optimierung durch den genetischen Algorithmus zu erreichen. Dadurch wäre jedoch nicht mehr die Optimalität der gefundenen Lösung garantiert, da diese im wesentlichen durch die Kompaktierung bestimmt würde. Hier soll während der Optimierung durch den genetischen Algorithmus mit einer schärferen Schranke für die benötigte Kanalbreite gearbeitet werden, welche möglichst genau den Verdrahtungsplatz abschätzt, den ein Kanalverdrahtungsverfahren benötigt, was im Anschluß an den Optimierungslauf die detaillierte Verdrahtung durchführt. Der exakte Verdrahtungsplatz kann auch schon während der Optimierung berechnet werden, jedoch ist es sinnvoller, hier weiterhin mit einer Abschätzung zu arbeiten, um die – ohnehin schon hohe – Laufzeit des genetischen Algorithmus zu reduzieren.

Eine weitere Schwäche der aktuellen Implementation liegt in der Tatsache, daß durch die hierarchische Konstruktion viele Netze nach außen geführt werden, welche anschließend über den Außenrand des Layouts zusammengeführt werden müssen. Um kürzere Verdrahtungswege zu erzeugen, ist es geplant, zusätzlich zur Layout-Fläche auch die Gesamtverdrahtungslänge bei der Berechnung der Fitneß zu berücksichtigen, so daß Individuen, welche Layouts mit kurzen Verdrahtungswegen repräsentieren, während der Optimierung eine größere „Überlebenschance" haben.

Aufgrund der natürlichen Parallelität der genetischen Algorithmen bietet sich eine Parallelisierung, selbst auf einem massiv parallelen System an, welche ebenfalls im Rahmen dieses Projektes durchgeführt werden soll.

Literatur

[1] J. P. Cohoon, W. D. Paris, *Genetic Placement*, Proc. of IEEE Int. Conf. on CAD 1986, 422–425

[2] J. P. Cohoon, S. U. Hegde, W. N. Martin, D. S. Richards, *Distributed Genetic Algorithms for the Floorplan Design Problem*, IEEE Trans. on CAD, Vol. 10 (4), April 1991, 483–492

[3] Ch. Darwin, *Die Entstehung der Arten durch natürliche Zuchtwahl*, Reclam, Stuttgart (1980)

[4] H. Esbensen, *A Macro-Cell Global Router based on two Genetic Algorithms*, Proc. of European Design Automation Conf. Euro-DAC 1994, Grenoble, France, 19.-23. September 1994, 428–433

[5] J. Lienig, K. Thulasiraman, *A Genetic Algorithm for Channel Routing in VLSI Circuits*, Evolutionary Computation, Volume 1, Number 4 (1994), 293–311

[6] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd Edition, Springer Verlag, 1994

[7] R. Otten, *Efficient Floorplan Optimization*, Proc. of Int. Conf. on Comp. Design 1983, 499–502

[8] S. M. Sait, H. Youssef, *VLSI Physical Design Automation: Theory and Practice*, McGraw-Hill (1995)

[9] V. Schnecke, O. Vornberger, *Genetic design of VLSI-layouts*, Procs. GALESIA '95, First IEE/IEEE Int. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications, 12.-14. September 1995, Sheffield, U.K.

[10] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, 1993